**This program is composed of 4 Python files:**

**This program consists of the following four Python files:**

1. **extract.py:  
   Serves as the main orchestrator for the entire pipeline, including tasks like web crawling, indexing, clustering, and sentiment analysis. It also handles tf-idf ranking, document clustering, and sentiment scoring. The crawling logic is based on a custom script tailored for the project.**
2. **inverted\_index.py:  
   Implements a custom web crawler designed to extract content from the Concordia Open Access portal, specifically targeting thesis documents. The variables, inputs, and functions are optimized to seamlessly integrate into the overall workflow.**
3. **cluster.py:  
   Contains all global variables and shared helper methods that are utilized by both main.py and crawler.py.**
4. **test.py:  
   Includes additional test cases created to verify PDF-related functionalities and ensure proper handling during the pipeline execution.**

**Output Files**

**The program generates the following output files:**

* **clusters\_3.png: A graphical representation of the clustering results for 3 clusters.**
* **clusters\_6.png: A graphical representation of the clustering results for 6 clusters.**
* **clustering\_6\_clusters.txt: A text file summarizing the clustering information for 6 clusters.**
* **clustering\_3\_clusters.txt: A text file summarizing the clustering information for 3 clusters.**
* **inverted\_index.txt: The generated inverted positional index for the corpus.**
* **top\_terms\_3\_clusters.txt: A file containing the top 20 terms from each of the 3 clusters with their tf-idf weights.**
* **top\_terms\_6\_clusters.txt: A file containing the top 20 terms from each of the 6 clusters with their tf-idf weights.**

**Execution Instructions**

**Before starting the project just run the requiremtns,txt install all of them.**

**Dependencies and Installation**

**Install the required Python packages using the provided requirements.txt.**

1. **Create a virtual environment:**
   * **UNIX/MacOS: python3 -m venv env**
   * **Windows: py -m venv env**
2. **Activate the virtual environment:**
   * **UNIX/MacOS: source env/bin/activate**
   * **Windows: .\env\Scripts\activate**
3. **Install dependencies:**

**pip install -r requirements.txt**

**To execute the project, follow these steps:**

**1. First naviaget to extract.py and tehn you can run with the command :**

**Python3 extract.py**

**A computer screen with blue text

Description automatically generated**

**I have set int eh cofde to use 5 by default if you wan tot increasr this use the command**

**Python3 extract.py --max\_files 10**

**This will run with 10 files from each faculty.**

1. **Sample Prompts at Execution:**
   * **"Enter the maximum number of files to scrape and download (default: 50):"**
2. **Configuration File Variables:**
   * **Max\_files: Maximum number of pages to crawl.**
   * **OUTPUT\_DIR: The directory where downloaded PDFs will be saved.**

**2. Run the inverted\_index.py using the command**

**Python3 inverted\_index.py**

**After running this, ths will do the inverted indexinf to the file inverterde\_index.txt.**

**A screenshot of a computer

Description automatically generated**

**The output file will be saved at inverted\_index.txt**

**A screen shot of a computer screen

Description automatically generated**

**3. Aftre rethe invetered\_index is generated then run the cluster.py code**

**You can run it with :**

**Python3 cluster.py**

**A screenshot of a computer program

Description automatically generated**

**My Experience and Challenges**

It took a considerable amount of time to identify the correct webpage to download the PDFs. Coding the scraper was an even more challenging task. I encountered rate limit errors and had to make numerous adjustments to the code to reduce the number of searches performed on the website.

Through experimentation, I discovered that the Spectrum website has a rate limit of approximately 130 requests, after which it blocks the IP address to prevent potential DoS attacks. To avoid issues, I set a default limit of 50 downloads per faculty, ensuring the scraping process would not violate the website’s restrictions.

Finding the correct PDF links was particularly tough due to frequent redirections and the presence of quick links at the top of each page. I had to identify the correct <a href> tag that pointed to the actual document and skip irrelevant links, such as “HowtoPrepareYourThesisForDepositinSpectrum.pdf,” which appeared on every page. Once I managed to extract and download the required documents, the process became smoother.

Before removing stopwords, the inverted index document had a total size of 180 MB. By using a standard list of 150 stopwords, I reduced the size by 26%, resulting in more efficient document indexing and improved TF-IDF scores.

Building the inverted index was relatively straightforward, but clustering posed another set of challenges. The code was incorrectly treating document names as words. For example:

plethora: [('Novakova\_PhD\_F2018', [7621]), ('Joyce\_PhD\_F2016', [2374]), ('Allen\_MA\_F2015', [3342]), ('Gupta\_MCompSc\_F2019', [2606]), ('Dinh\_MASc\_S2020', [197]), ('Olivera-Aravena\_MSc\_S2017', [4441])]

While "plethora" occurred only once, document names like "Novakova\_PhD\_F2018" were being treated as words and appeared for every single word in the corresponding document. To address this, I wrote a loop to iterate through all the departments and collect document names into a list. Using a union-find method, I ensured that both stopwords and document names were filtered out during preprocessing.

Thois eliminated that problem, and the clustering is done with 3 clusters and 6 clusrters and top 20 words of each cluster have been put into the clustering\_3\_clusters.txt and clustering\_6\_clusters.txt . the basic fomat of it is :

Cluster 1:  
https: 0.1341  
open: 0.1279  
wikidata: 0.1062  
org: 0.1041  
oa: 0.0769  
library: 0.0724  
2023: 0.0722  
visualization: 0.0719  
de: 0.0669

And the cluster images have been saved successfully\

Finally